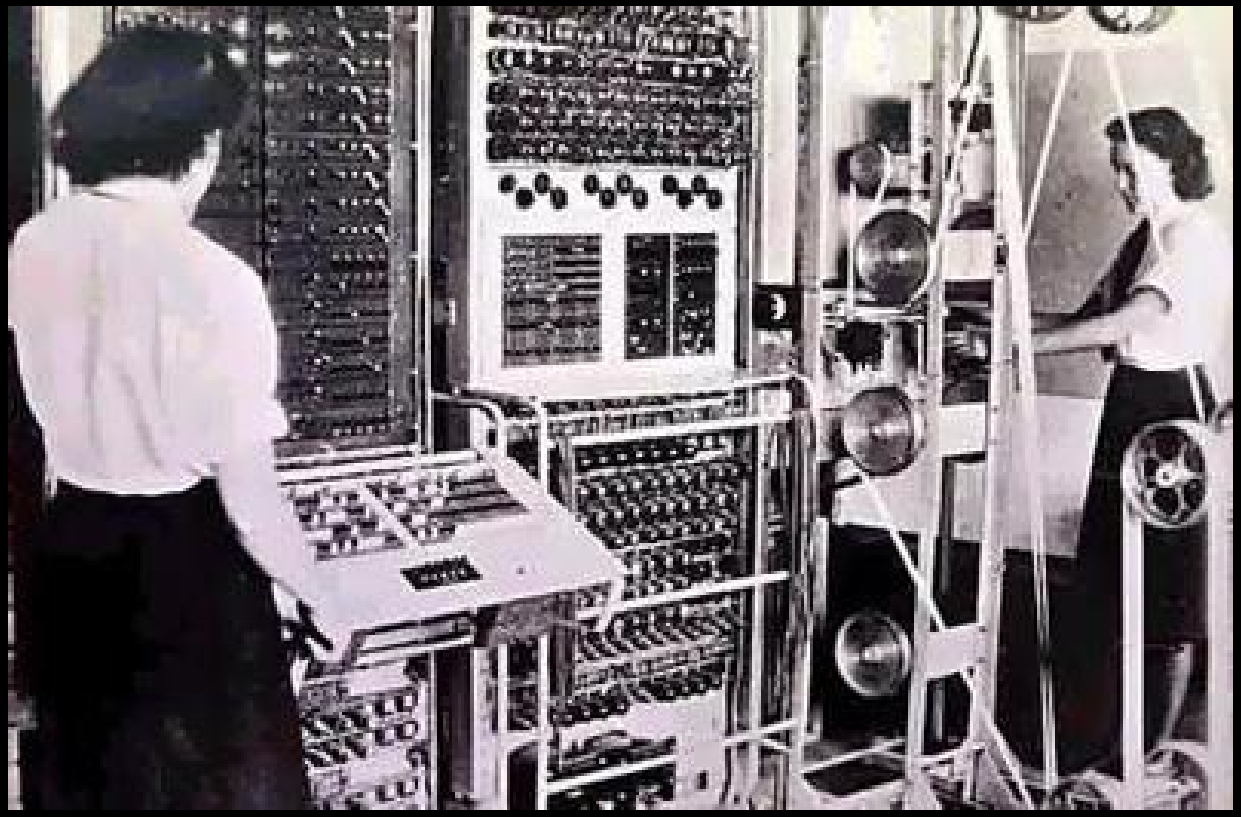
Numba

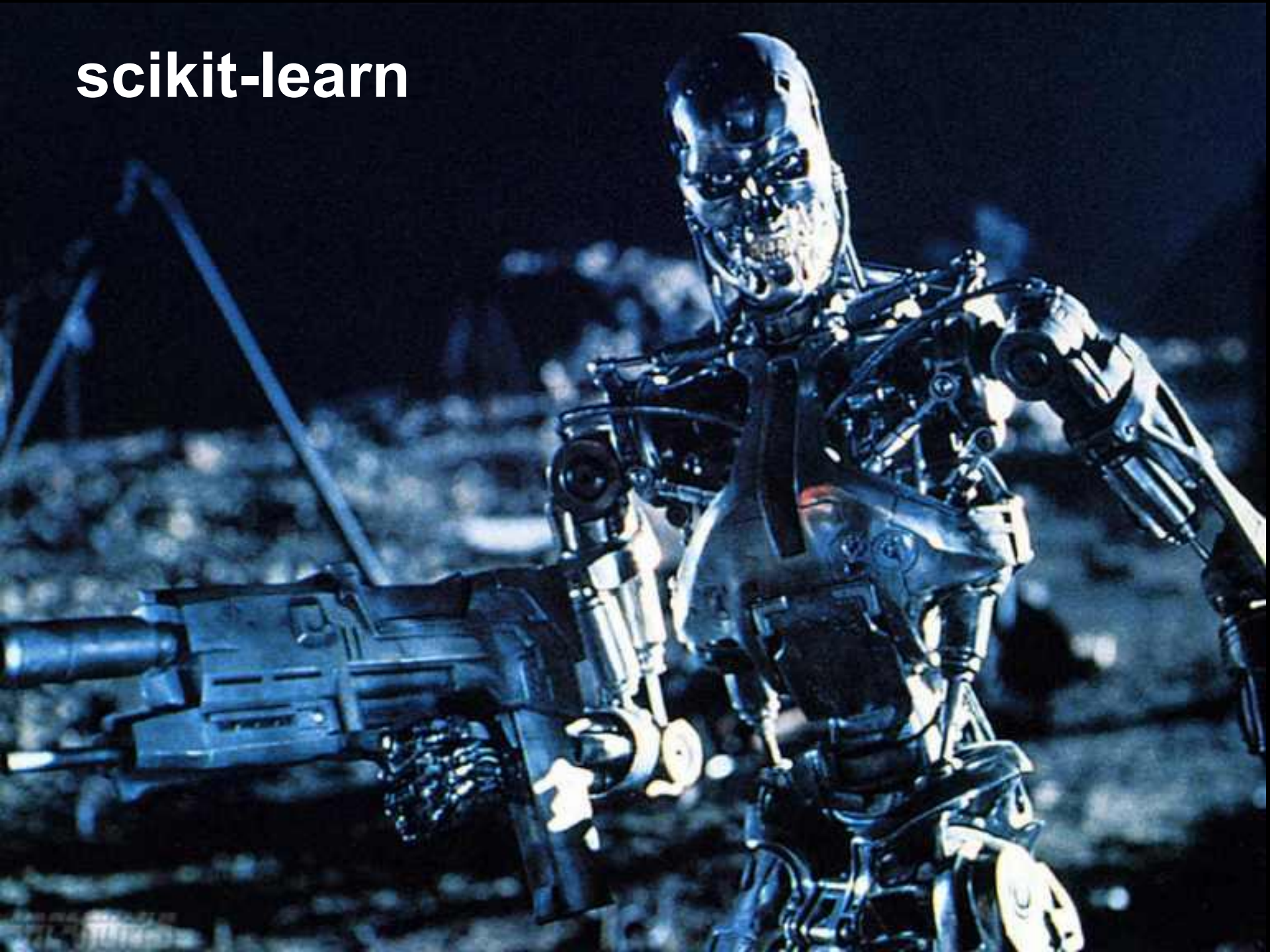
Fast loops without without Compilation

Fast Loops!

Why?
Scientific Computing / Number crunching



scikit-learn



scikit-image

Cython : Fast loops with compiling

1. Write Python code
2. Compile to C with Cython
3. Optional strong typing for speed

LOGICAL AWESOME

My Codes Are Perfect

BUT pretty static :-/

Enter Numba

Travis Oliphant's new project
http://numba.pydata.org

LLVM JIT via decorators!

Tight Numpy integration

```
import numpy as np

def pairwise_python(X, D):
    M = X.shape[0]
    N = X.shape[1]
    for i in range(M):
        for j in range(M):
            d = 0.0
            for k in range(N):
                tmp = X[i, k] - X[j, k]
                d += tmp * tmp
            D[i, j] = np.sqrt(d)
```

```
1 loops, best of 3: 12.1 s per loop
```

```
import numpy as np
from numba import double
from numba.decorators import jit

@jit(arg_types=[double[:,:], double[:,:]])
def pairwise_numba(X, D):
    M = X.shape[0]
    N = X.shape[1]
    for i in range(M):
        for j in range(M):
            d = 0.0
            for k in range(N):
                tmp = X[i, k] - X[j, k]
                d += tmp * tmp
            D[i, j] = np.sqrt(d)
```

```
100 loops, best of 3: 15.5 ms per loop
```

```
1   cimport cython
2   from libc.math cimport sqrt
3
4   @cython.boundscheck(False)
5   @cython.wraparound(False)
6   def pairwise_cython(double[:, ::1] X, double[:, ::1] D):
7       cdef int M = X.shape[0]
8       cdef int N = X.shape[1]
9       cdef double tmp, d
10      for i in range(M):
11          for j in range(M):
12              d = 0.0
13              for k in range(N):
14                  tmp = X[i, k] - X[j, k]
15                  d += tmp * tmp
16              D[i, j] = sqrt(d)
```

```
5   100 loops, best of 3: 9.86 ms per loop
```

Example stolen from Jake:
http://jakevdp.github.com

Resources

Website
http://numba.pydata.org

Github
https://github.com/numba/numba

Early stage, worth taking a look!